ERDC MSRC PET Technical Report No. 01-12

Library of Grid Interpolation Modules (INLib)

by

S. Gopalsamy
Bharat K. Soni

4 May 2001

Work funded by the Department of Defense
High Performance Computing Modernization Program
U.S. Army Engineer Research and Development Center
Major Shared Resource Center through

Programming Environment and Training

Supported by Contract Number: DAHC94-96-C0002
Computer Sciences Corporation

Library of Grid Interpolation Modules (INLib)

S. Gopalsamy[1] Bharat K. Soni[2]

1 Introduction

Interpolation of data/solution values from one grid to another, known as *grid interpolation*, is required in many applications dealing with dynamically moving/deforming/changing grids and also in multidisciplinary applications requiring distinct grid resolution/strategies. Grid manipulation tools, including grid interpolation, are utilized at many DoD simulation steps starting from geometry preparation and grid generation to post-processing and visualization. These steps are associated with numerical simulations involving Computational Fluid Dynamics (CFD), Computational Structural Mechanics (CSM), and other Computational Technology Areas.

Important tasks in grid interpolation are: (a) finding a grid cell containing a given point and (b) computing interpolation parameters of a point with respect to a grid cell. Since the grids can be very large, it is required that these two tasks are performed very efficiently. Various algorithms and software components are available in the literature and in the public domain to perform grid interpolation in a piecemeal fashion. Researchers are also exploring distinct interpolation algorithms within the academic/NASA/DoD community. However, there is no coherent library/tool kit accessible to the DoD community to provide required functionalities. In response to this need the development of INLib (Interpolation Library) was initiated as a collaborative effort between Mississippi State University and the University of Texas at Austin.

An alpha version of INLib has been completed. It contains stand-alone interpolation functions for the following types of grids: multi-block structured volume grids consisting of hexahedrons, unstructured volume grids consisting of tetrahedrons, and prismatic volume grids consisting of prism elements. The library functions are written in C and Application Programming Interfaces (APIs) are available in C and FORTRAN.

INLib is being developed as one of the modules of GGTK, which is an extensive library of geometry and grid functions currently under development [1]. Figure 1 shows some of the modules in GGTK. The plan is to extract any desired and available functionalities from existing code bases, add new functionalities wherever necessary, integrate all the functionalities into libraries, and provide C and FORTRAN APIs. The approach is as follows:

1. Identify various modules to be included in GGTK such as NURBS, structured and unstructured grids, grid interpolation, elliptic grids, grid quality, transfinite interpolation, etc.

2. For each module, extract functions from existing code bases like GGLib and Genie. The functions may be in FORTRAN or C.

[1]Visiting senior research assistant, Center for Computational Systems, ERC, Mississippi State University
[2]Professor, Aerospace Engineering, Center for Computational Systems, ERC, Mississippi State University, PET Academic CFD Lead

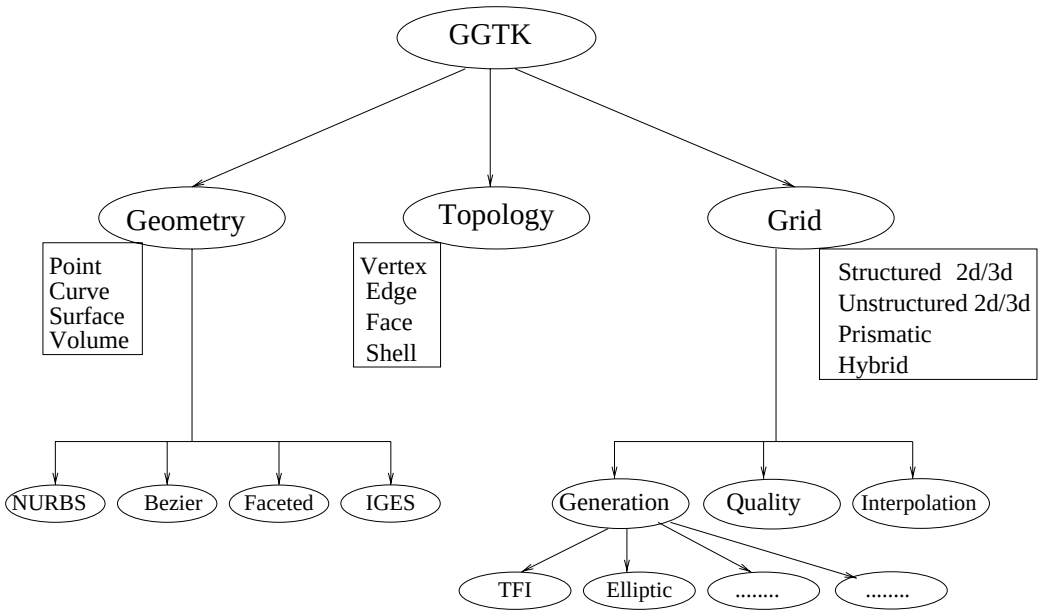

Figure 1: Currently existing modules in GGTK

3. Add new functions as required in various modules.

4. Identify common geometry and grid entities and define common data structures in C. This procedure will help to provide uniform APIs in C and will also help in defining common parameters for entities across various functions for FORTRAN APIs.

5. Compile the functions in various modules, creating libraries and providing ways of linking the libraries to other applications.

6. Develop and test APIs in C and FORTRAN.

In this report, a detailed description of the grid interpolation module is given. Descriptions of various entities and data-structures associated with grid interpolation are given first followed by the main functionality of the module. Next, implementation details and brief descriptions of some of the algorithms used are provided. Then, results of experiments on some test data are given. Finally, conclusions and work in progress are reported.

2 Entities and Data-structures

Currently INLib contains functions for grid interpolation of three types of grids: multi-block structured volume grids, unstructured volume grids, and prismatic volume grids. The main entities identified by their C structure names are:

- Basic 3D entities: Point3d or point_t, Tetrahedron, and Box3d
- 3D Grid entities: StructGrid, MBStructGrid, UnstructGrid, and PrismaticGrid
- Utility entities: Solution, and Octrees of bounding boxes.

2.1 Basic 3D Entities

C data-structures of the basic 3d entities are given below.

```
typedef struct {
    double x, y, z;
} point_t, Point3d, Vector3d;

typedef struct tetrahedron_t {
    point_t     vert[4];
} Tetrahedron;

typedef struct Box3d_t {
    point_t     min;
    point_t     max;
} Box3d;
```

2.2 3D Grid Entities

C data-structures of the 3d grid entities are given below.

```
/* 3D Structured Grid entities */

typedef struct structured_grid_t {
    int         ni;         /* number of points in i-direction */
    int         nj;         /* number of points in j-direction */
    int         nk;         /* number of points in k-direction */
    point_t*    points;     /* Pointer to array of points */
} StructGrid;

typedef struct multi-block_structured_grid_t {
    int             nblocks;    /* number of blocks */
    StructGrid*     stgrids;     /* Pointer to array of structured grids */
} MBStructGrid;

/* 3D Unstructured Grid entities */

typedef struct tetrahedron_cell {
    int vert[4];
} Tetcell;

typedef struct cell_neighbors {
    int nbr[4];
} Nbrcells;

typedef struct boundary_triangle {
```

```
    int vert[3];
    int flag;       /* boundary flag */
} Bdtri;

typedef struct unstructured_grid {
    int         npoints, nbdtris, ntets;
    point_t     *points;
    Bdtri       *bdtris;
    Tetcell     *tets;
    Nbrcells    *nbrs;
} UnstructGrid;

/* 3D Prismatic Grid entities */

typedef struct prism_cell {
    int vert[6];
} Prmcell;

typedef struct prism_cell_neighbors {
    int nbr[5];
} PRGNbrcells;

typedef struct boundary_element {
    int nvert;
    int vert[4];
    int flag;       /* boundary flag */
} Bdelm;

typedef struct prismatic_grid {
    int         npoints, nbdelms, nprms;
    point_t     *points;
    Bdelm       *bdelms;
    Prmcell     *prms;
    PRGNbrcells *nbrs;
} PrismaticGrid;
```

2.3 Utility Entities

C data-structures of the utility entities are given below.

```
typedef struct solution_for_a_grid_t {
    char    solutionLocations[2];   /* To specify solution values are at    */
                                    /* grid points or at cell centers       */
    int     nvalues;                /* Number of values */
    double* values;                 /* Solution values  */
} Solution;
```

```
/* Octree of bounding boxes of a Structured Grid  */

typedef struct octreeNode_t OctreeNode;
typedef struct octreeNode_t {
    int         i0, i1;     /* bounds of indices in i direction */
    int         j0, j1;     /* bounds of indices in j direction */
    int         k0, k1;     /* bounds of indices in k direction */
    Box3d       bdbox;
    int         numChildren;
    OctreeNode**    children;
} Octree;

/* Octree of bounding boxes for Unstructured and Prismatic grids */

typedef struct bbx_Octree_Node bbxOctreeNode;
struct bbx_Octree_Node {
    arrayInts       *bucket;
    Box3d           *bdbox;
    bbxOctreeNode   **children;      };
```

3 Functions and APIs

For each of the structured, unstructured, and prismatic volume grids, the following set of main and utility functions are given:

1. Main grid interpolation function for a given grid and solution;
2. Finding a cell (hex or tet or prism) containing a given point;
3. Computation of interpolation parameters of a point w.r.t. a cell – u, v, w for hex cells and $\alpha, \beta, \gamma, \delta$ for tet or prism cells;
4. Computation of octrees, which are used in search routines, for structured, unstructured and prismatic volume grid.

A complete list of API functions in *C syntax* is given in the Appendix.

4 Implementation Details

Brief details of the algorithms used for search and interpolation tasks are given below with respect to the three types of grids – 3D Structured grids consisting of hexahedrons, 3D unstructured grids consisting of tetrahedrons, and 3D prismatic grids consisting of six-noded prisms.

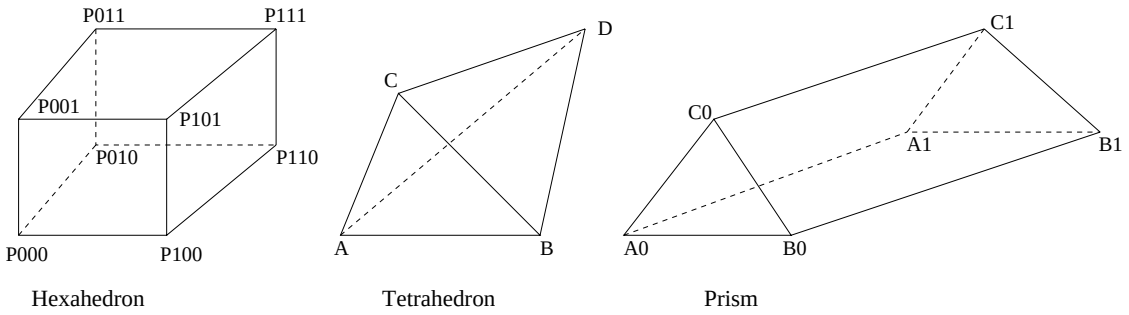

Figure 2: Three kinds of 3D grid cells

4.1 Interpolation Parameters

Methods for computation of interpolation parameters of a point with respect to the three kinds of grid cells, shown in Figure 2, are given below.

4.1.1 Interpolation Parameters with respect to a Hexahedron

Let P be a point in $3D$ and H be a hexahedron with vertices P_{000}, P_{001}, P_{010}, P_{011}, P_{100}, P_{101}, P_{110}, P_{111} (see Figure 2); then the interpolation parameters u, v, w of P with respect to H are defined by the following equation:

$$\begin{aligned} P = (1-u)(1-v)(1-w)P_{000} + (1-u)(1-v)(1+w)P_{001} + \\ (1-u)(1+v)(1-w)P_{010} + (1-u)(1+v)(1+w)P_{011} + \\ (1+u)(1-v)(1-w)P_{100} + (1+u)(1-v)(1+w)P_{101} + \\ (1+u)(1+v)(1-w)P_{110} + (1+u)(1+v)(1+w)P_{111} \end{aligned} \quad (1)$$

For given P and H, this is a non-linear equation in u, v, w and can be solved numerically by the Newton-Raphson iterative method [2]. The procedure given in [2] has been implemented and the method has been found to be efficient. For a non-degenerate hexahedron, the method converges within ten iterations.

The point lies inside or on the hexahedron if $-1 \leq u, v, w \leq 1$.

4.1.2 Interpolation Parameters with respect to a Tetrahedron

Let P be a point in $3D$ and T be a tetrahedron with vertices A, B, C, D. Then the interpolation parameters $\alpha, \beta, \gamma, \delta$ of P with respect to T are defined by the following equations:

$$P = \alpha A + \beta B + \gamma C + \delta D \quad (2)$$

$$\alpha + \beta + \gamma + \delta = 1 \quad (3)$$

This results in four linear equations (in $\alpha, \beta, \gamma, \delta$) that can be solved exactly for any non-degenerate tetrahedron.

The point lies inside or on the tetrahedron if $0 \leq \alpha, \beta, \gamma, \delta \leq 1$.

4.1.3 Interpolation Parameters with respect to a Prism

Let P be a point in $3D$ and S be a six-noded prism with vertices $A_0, B_0, C_0, A_1, B_1, C_1$ (see Figure 2); then the interpolation parameters $\alpha, \beta, \gamma, \delta$ of P with respect to S are defined by the following equations:

$$P = (1-\delta)(\alpha A_0 + \beta B_0 + \gamma C_0) + \delta(\alpha A_1 + \beta B_1 + \gamma C_1) \tag{4}$$

$$\alpha + \beta + \gamma = 1 \tag{5}$$

Using equation (5) we can write $\gamma = 1 - \alpha - \beta$. By substituting this for γ in equation (4), we can eliminate γ and obtain a single vector equation in α, β, δ as follows:

$$P = (1-\delta)[\alpha(A_0 - C_0) + \beta(B_0 - C_0) + C_0] + \delta[\alpha(A_1 - C_1) + \beta(B_1 - C_1) + C_1] \tag{6}$$

This equation is a non-linear equation (in α, β, δ) that can be solved numerically by the Newton-Raphson iterative method, in the same way that we solve equation (1) for u, v, w.

The point lies inside or on the prism if $0 \leq \alpha, \beta, \gamma, \delta \leq 1$.

4.2 Search Methods

Given a grid and a point P, the task of a search method is to determine if the point lies in the grid and, in that case, to find the grid cell containing the point. For structured and unstructured grids, two kinds of search methods have been implemented. They are search by traversal and search by octree. For prismatic grids only search by octree has been implemented.

4.2.1 Search by Traversal

Search by traversal is done as follows (see reference [2] for complete details):

1. Choose an initial trial cell of the grid.
2. Find interpolation parameters of the point with respect to the cell. Mark the current trial cell as visited.
3. Using the interpolation parameters, determine whether the point lies in the cell or not.
4. If the point lies in the cell, the search is over - RETURN the cell id and the interpolation parameters.
5. If not, select the neighboring cell as determined by the interpolation parameters.
6. If the neighboring cell exists, take that as the trial cell and go to Step 2 above.
7. If the neighboring cell does not exist, it indicates that a boundary has been hit. Go to either Step 8 or Step 11.
8. Choose a cell that has not been marked as visited.

9. If such a cell exists, take that as the trial cell and go to Step 2 above.

10. If there is no unvisited cell, RETURN saying that the search is over and the point does not lie in the grid.

11. Halt if the search reaches a boundary. RETURN saying that the search has failed. In this case one can call another search method, such as the octree search, and find the point.

From the above procedure, it can be seen that efficiency of the traversal method depends on the initial choice of a trial cell. If a sequence of points is being searched for, the returned cell of the previous point is taken as the initial trial cell of the current point. For the first point, one generally takes the first grid cell as the initial trial cell. In our implementation, the cell for the first point is found by the octree method.

Secondly, if a boundary is hit (Step 7), it indicates that the point is outside the grid or that the domain of the grid is concave. In the case of concave domains, it may be time consuming to continue the search with another unvisited trial cell. In our implementation, the traversal in that case is abandoned (Step 11) and the point is searched by the octree method.

4.2.2 Search by Octree

In this method an octree of bounding boxes of the grid is constructed as a pre-processing step. The bounding box of the whole grid is taken as the starting (root) bounding box. It is recursively subdivided into smaller bounding boxes. Each bounding box is associated with a bucket which stores the identifiers of all of the cells which intersect the bounding box. If a bucket is empty or contains less than a predefined minimum number of cells then it corresponds to a leaf node. Search using an octree is done as follows:

1. Take the root as the current octree node.

2. Check if the given point lies in the bounding box of the current node.

3. If the point does not lie in the bounding box, return not found.

4. If the point lies in the bounding box, and if it is a leaf node, go to Step 5; else go to Step 6.

5. For each of the cells in the bucket of the leaf node, check if the point lies in the cell by computing interpolation parameters of the point. If the point lies in any of the cell, return found along with the cell id. If not, return not found.

6. Identify the child octant the point lies in. Take that octant as the current node and go to Step 2.

5 Experimental Results

The grid interpolation functions of INLib have been tested and found to be working correctly for the test data. The results of our experiments with three grids, one each for structured, unstructured, and prismatic grids, are given below. In each of these cases, solution values at the vertices of the grid were generated using analytical solution functions.

5.1 Structured Grid

The grids shown in Figure 3 were used to test structured grid interpolation functions. Both were 3D structured grids of the same size ($30 \times 20 \times 10$ in I, J, K directions) over the same volume. The grid on the left, with highly skewed hexahedrons, was taken as the given grid and the uniform grid on the right was taken as the target grid. Search and interpolation of solution values were done for the vertices of the target grid and the following results were obtained:

- Pre-computation of the Octree of bounding boxes of the given grid did not take much time. It was only a fraction of the search time and for such a small grid it was negligible.
- Out of 6000 points (vertices) of the target grid, 478 points were not found to be inside the given grid. They were actually outside the given grid for the tolerance value of 0.01, which was used for inside/outside test using interpolation parameters u, v, w. When the tolerance value was increased to 0.5, all the points were inside.
- The interpolated solution values were compared with the values computed directly from the solution function. The results were as expected. The solution values were exact (up to machine tolerance) if the solution function was a linear polynomial. For non-linear solution functions they were not as exact because the interpolation was only linear inside each grid cell.

5.2 Unstructured Grid

Unstructured grid interpolation functions were tested with a grid (shown in Figure 4) having 166,139 vertices, 76,308 boundary triangles, and 873,256 tetrahedrons. It is a volume grid used to compute wind-flow simulation over a small part of a city. For this grid, the pre-computation of the Octree of bounding boxes took about four minutes. The interpolated solution values were exact for linear solution functions.

5.3 Prismatic grid

To test the prismatic grid interpolation functions, we converted the structured grid shown in the left half of Figure 3, into a prismatic grid by splitting each hexahedron into two prisms as shown in Figure 5. Here again, the computation of the octree of bounding boxes was fast and the interpolation was exact for linear solution functions.

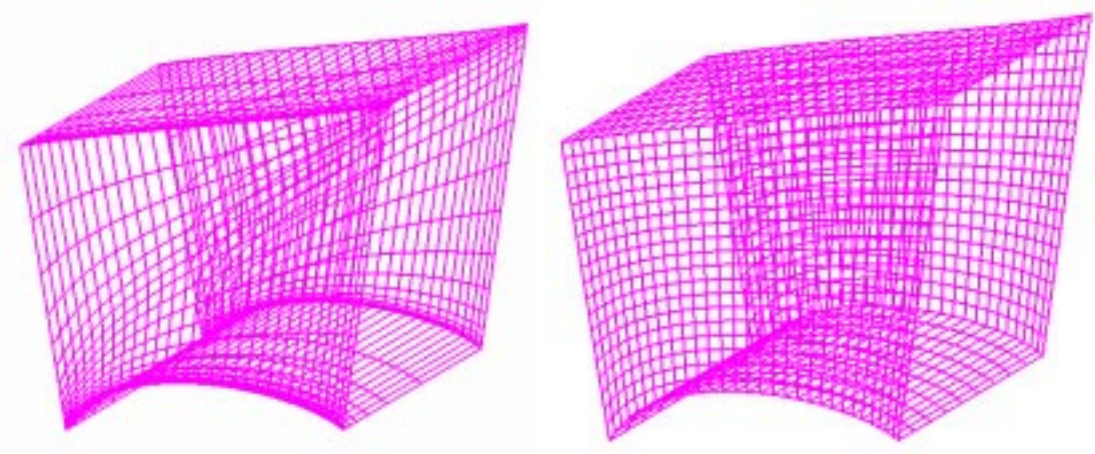

Figure 3: Given (left) and target (right) 3D structured grids of size $30 \times 20 \times 10$ over same volume.

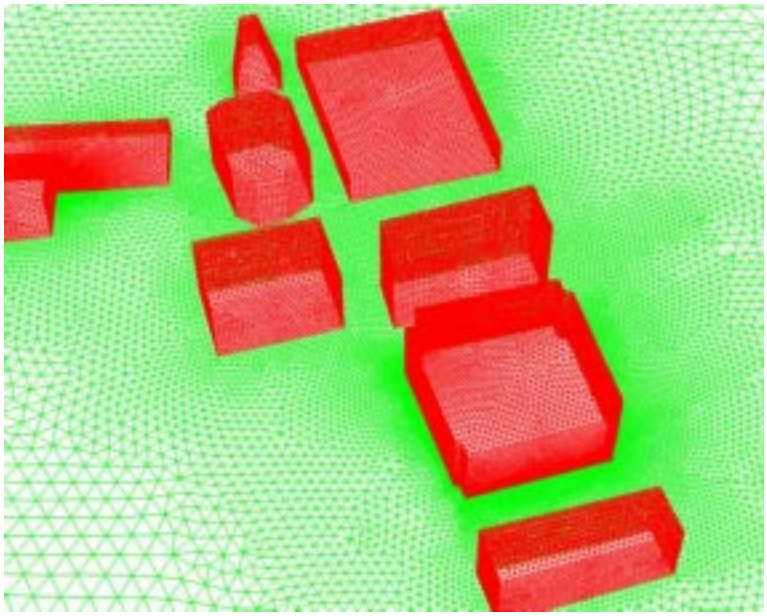

Figure 4: Unstructured grid over a few buildings of a city

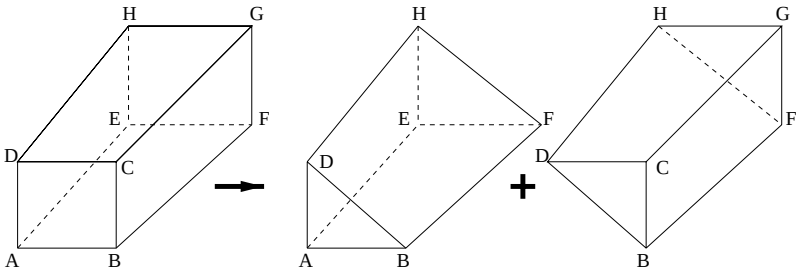

Figure 5: Splitting a hexahedron into two prisms

6 Conclusions and Work in Progress

Stand alone grid interpolation functions have been developed for structured, unstructured, and prismatic volume grids in the form of a library of functions. The functions are coded in C, and C data-structures are given for all of the associated entities. The functions have been tested using a few sample grids. The results showed that the interpolated values were exact for linear solution functions. They also showed that the pre-computation of the octrees used in the search methods took only a fraction of the total search time.

In order that the library may be widely used, the following tasks have been identified and the implementation work is in progress:

- Test and demonstrate INLib functions using data from actual applications.
- Incorporate grid interpolation for generalized volume grids consisting of mixed volume cells.
- Experiment with higher order interpolation functions.

References

[1] B. Soni, D. Thompson, R. Koomullil, and H. Thornburg. Ggtk: A tool kit for static and dynamic geometry-grid generation and adaptation. In *AIAA 2001-1164, 39th Aerospace Sciences Meeting &Exhibit, 8-11 January 2001, Reno, Nevada.*

[2] M. L. Stokes and K. R. Kneile. A new search/interpolation scheme for structured and unstructured grid systems with applications in computational fluid dynamics. In *Proceedings from the First World Congress on Computational Mechanics, Austin, TX*, pages 22–25, September 1986.

Appendix: List of INLib Functions

Read and write functions

```
MBStructGrid* readMBStructGrid(char* filename); /* Fast3d format */
MBStructGrid* readMBStructGridPoint3dFormat(char* filename);
void writeMBStructGrid(MBStructGrid* mbgrid, char* filename);

UnstructGrid* usgReadUnstructGrid(FILE* gridfile);
void usgWriteUnstructGrid(UnstructGrid* grid, FILE* gridfile);

PrismaticGrid* prgReadPrismaticGrid(FILE* gridfile);
void prgWritePrismaticGrid(PrismaticGrid* grid, FILE* gridfile);
```

3D Structured grid interpolation functions

```
int MBStructGridInterpolation(
                    MBStructGrid *givenGrid, Solution *givenSolution,
                    MBStructGrid *targetGrid, Solution **targetSolutionPtr);

OctreeNode* buildStructGridOctree(StructGrid* grid, int i0, int i1,
                                  int j0, int j1, int k0, int k1);

int SGOctreeSearch(point_t* pt, StructGrid* grid, OctreeNode* octree,
                         int *icell, int *jcell, int *kcell,
                         double* alpha, double* beta, double* gamma,
                         double tol1, double tol2);

void getHexCellVertices(StructGrid* grid, int i0, int j0, int k0,
                double x[8], double y[8], double z[8]);

int CheckHexa(double *xyz,
              double x[],double y[],double z[],
              double *alpha, double *beta, double *gamma,
              double tol1,double tol2);

int find_3d_cell(point_t* pt, StructGrid* grid,
                 int *i1,int *j1,int *k1,
                 double *a1,double *b1, double *g1,
                 double tol1, double tol2);

double interpolateValues(MBStructGrid* givenGrid,
                         Solution* givenSolution,
                         int block, int icell, int jcell, int kcell,
                         double alpha, double beta, double gamma);
```

3D Unstructured grid interpolation functions

```
int usgUnstructGridInterpolation(UnstructGrid* grid,
                                 arrayDoubles *givenSolution,
                                 int npts, point_t* pts,
                                 arrayDoubles **targetSolutionPtr);

int usgFindCell(UnstructGrid* grid, point_t* pt,
                int* currentcell, double* Cf, double Tol);

int usgFind_Coeffs(point_t* fpoint, Tetrahedron* current_cell,
                 double* Coeffs);

usgOctreeNode* usgCompOctreeNode(UnstructGrid* grid, Box3d* bdboxCells,
                Box3d* nodeBdbox, arrayInts* bucket,
                int maxsize, int currentLevel, int maxLevel);

int usgFindCellByOctree(UnstructGrid* grid, point_t* pt,
                        usgOctreeNode* octree,
                        Box3d* gridBdbox, Box3d* cellBdboxes,
                        int* cellid, double* Cf, double Tol);

double usgInterpolateValues(UnstructGrid* grid, arrayDoubles *soln,
                            int cellid, double* cf);
```

3D Prismatic grid interpolation functions

```
int prgPrismaticGridInterpolation(PrismaticGrid* grid,
                                  arrayDoubles *givenSolution,
                                  int npts, point_t* pts,
                                  arrayDoubles **targetSolutionPtr);

bbxOctreeNode* prgCompOctreeofPrms(PrismaticGrid* grid, Box3d* gridBdbox,
                                   Box3d** bdboxCells, double tol);

int prgFindCellByOctree(PrismaticGrid* grid, point_t* pt,
                        bbxOctreeNode* octree,
                        Box3d* gridBdbox, Box3d* cellBdboxes,
                        int* cellid, double* Cf, double Tol);

int prgPrismParametersOfPoint(double* x, double* y, double* z, point_t* pt,
              double* Cf, double tol);

double prgInterpolateValues(PrismaticGrid* grid, arrayDoubles *soln,
                            int cellid, double* cf);
```